Chiem Exercise 8

MAT 115
Exercise #8

In this exercise we are going to continue exploring the tidyverse (sections 4.5 to 4.14 of the text). We will mostly be focusing on the programming concept called piping.

This will be a two-class day exercise. Stop around line 150 on the first day and finish the rest on the second day.

As usual, we first load `dslabs` and `tidyverse`. We will use a new dataset for this exercise. It is the `starwars` dataset from the `dplyr` package (this seems appropriate, no?). Note, `tidyverse` is actually a set of packages and `dplyr` is part of that set, so you do not need to load it separately.

```
library(tidyverse)
library(dslabs)
```

4.5 How do we calculate the value of $\sin(\sqrt{e^x})$ when $x = 1.38629$? Well, we execute the following steps:

$$1.38629 \to e^{1.38629} = 4 \to \sqrt{4} = 2 \to \sin(2) = 0.91$$

The function notation $\sin(\sqrt{e^x})$ is in a sense backwards. When we plug in $x = 1.38629$, the first thing we do is take the exponential, not the sine. The sequence of calculation steps is more in keeping with how we humans think. The R pipe `%>%` is an attempt to implement this step-by-step way of thinking. When we write something like `LHS %>% RHS`, it means that the function on the RHS takes the LHS as its first argument. So `2 %>% exp` means the same thing as `exp(2)`, and `8 %>% log(2)` means the same thing as `log(8,2)`.

```
# Try out those commands.
2 %>% exp
```

```
## [1] 7.389056
```

```
#exp(2)

8 %>% log(2)
```

```
## [1] 3
```

```
#log(8, 2)
```

Recall that we can use the `mutate` function to constructed new dataframes. Here is an example:

```
newwars <- mutate(starwars,avHW=(height+mass)/2)
#newwars
```

This can also be written as

```
newwars <- starwars %>% mutate(avHW=(height+mass)/2)
head(newwars)
```

```
## # A tibble: 6 x 15
##   name      height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
## 1 Luke Sky~    172    77 blond      fair       blue            19   male  mascu~
## 2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu~
## 3 R2-D2         96    32 <NA>       white, bl~ red             33   none  mascu~
## 4 Darth Va~    202   136 none       white      yellow          41.9 male  mascu~
## 5 Leia Org~    150    49 brown      light      brown           19   fema~ femin~
## 6 Owen Lars    178   120 brown, gr~ light      blue            52   male  mascu~
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

Which of these two versions do you find more intuitive? Why?

(Note: with R version 4.1.0 and above, the pipe operator `|>` was included in base R. The `%>%` operator is older and part of the tidyverse. `%>%` and `|>` work in very similar ways.)

4.9 If you want to reorder the rows of a dataframe, you can use the `arrange` function. For example, if you want to sort the star wars characters based on average height and weight , you can do this:

```
newwars %>% arrange(avHW)
```

```
## # A tibble: 87 x 15
##    name     height  mass hair_color skin_color eye_color birth_year sex   gender
##    <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
##  1 Yoda         66    17 white      green      brown            896 male  mascu~
##  2 Ratts T~     79    15 none       grey, blue unknown           NA male  mascu~
##  3 Wicket ~     88    20 brown      brown      brown              8 male  mascu~
##  4 R2-D2        96    32 <NA>       white, bl~ red               33 none  mascu~
##  5 R5-D4        97    32 <NA>       white, red red               NA none  mascu~
##  6 Dud Bolt     94    45 none       blue, grey yellow            NA male  mascu~
##  7 Sebulba     112    40 none       grey, red  orange            NA male  mascu~
##  8 Leia Or~    150    49 brown      light      brown             19 fema~ femin~
##  9 Barriss~    166    50 black      yellow     blue              40 fema~ femin~
## 10 Zam Wes~    168    55 blonde     fair, gre~ yellow            NA fema~ femin~
## # i 77 more rows
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

The sorting is done lowest to highest by default. Explore the help documentation of `arrange` and try to figure out how to sort highest to lowest.

```
# Write your code here.
newwars %>% arrange(desc(avHW));
```

```
## # A tibble: 87 x 15
##    name     height  mass hair_color skin_color eye_color birth_year sex   gender
##    <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
##  1 Jabba D~    175  1358 <NA>       green-tan~ orange         600   herm~ mascu~
##  2 Grievous    216   159 none       brown, wh~ green, y~       NA   male  mascu~
##  3 Tarfful     234   136 brown      brown      blue            NA   male  mascu~
##  4 Chewbac~    228   112 brown      unknown    blue           200   male  mascu~
##  5 IG-88       200   140 none       metal      red             15   none  mascu~
##  6 Darth V~    202   136 none       white      yellow          41.9 male  mascu~
##  7 Lama Su     229    88 none       grey       black           NA   male  mascu~
##  8 Roos Ta~    224    82 none       grey       orange          NA   male  mascu~
##  9 Bossk       190   113 none       green      red             53   male  mascu~
## 10 Dexter ~    198   102 none       brown      yellow          NA   male  mascu~
## # i 77 more rows
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

Reproduce this outcome with base R functionality.

```
# Write your code here.
newwars[order((newwars$height+newwars$mass)/2, decreasing = TRUE),]
```

```
## # A tibble: 87 x 15
##    name     height  mass hair_color skin_color eye_color birth_year sex   gender
##    <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
##  1 Jabba D~    175  1358 <NA>       green-tan~ orange         600   herm~ mascu~
##  2 Grievous    216   159 none       brown, wh~ green, y~       NA   male  mascu~
##  3 Tarfful     234   136 brown      brown      blue            NA   male  mascu~
##  4 Chewbac~    228   112 brown      unknown    blue           200   male  mascu~
##  5 IG-88       200   140 none       metal      red             15   none  mascu~
##  6 Darth V~    202   136 none       white      yellow          41.9 male  mascu~
##  7 Lama Su     229    88 none       grey       black           NA   male  mascu~
##  8 Roos Ta~    224    82 none       grey       orange          NA   male  mascu~
##  9 Bossk       190   113 none       green      red             53   male  mascu~
## 10 Dexter ~    198   102 none       brown      yellow          NA   male  mascu~
## # i 77 more rows
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

It is also possible to sort using two columns.

```
newwars %>% arrange(height, mass) %>% head(20)
```

```
## # A tibble: 20 x 15
##    name     height  mass hair_color skin_color eye_color birth_year sex   gender
##    <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
##  1 Yoda         66    17 white      green      brown            896 male  mascu~
##  2 Ratts T~     79    15 none       grey, blue unknown           NA male  mascu~
```

```
##  3 Wicket ~     88    20 brown      brown      brown             8 male  mascu~
##  4 Dud Bolt     94    45 none       blue, grey yellow           NA male  mascu~
##  5 R2-D2        96    32 <NA>       white, bl~ red              33 none  mascu~
##  6 R4-P17       96    NA none       silver, r~ red, blue        NA none  femin~
##  7 R5-D4        97    32 <NA>       white, red red              NA none  mascu~
##  8 Sebulba     112    40 none       grey, red  orange           NA male  mascu~
##  9 Gasgano     122    NA none       white, bl~ black            NA male  mascu~
## 10 Watto       137    NA black      blue, grey yellow           NA male  mascu~
## 11 Leia Or~    150    49 brown      light      brown            19 fema~ femin~
## 12 Mon Mot~    150    NA auburn     fair       blue             48 fema~ femin~
## 13 Cordé       157    NA brown      light      brown            NA <NA>  <NA>
## 14 Nien Nu~    160    68 none       grey       black            NA male  mascu~
## 15 Ben Qua~    163    65 none       grey, gre~ orange           NA male  mascu~
## 16 Shmi Sk~    163    NA black      fair       brown            72 fema~ femin~
## 17 Beru Wh~    165    75 brown      light      blue             47 fema~ femin~
## 18 Dormé       165    NA brown      light      brown            NA fema~ femin~
## 19 Barriss~    166    50 black      yellow     blue             40 fema~ femin~
## 20 C-3PO       167    75 <NA>       gold       yellow          112 none  mascu~
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

Inspect this output and explain in your own words how ***arrange*** *sorts by two columns. How does it handle NA values?*

A: Irregardless of mass, it seems to sort by height in ascending order, ignoring the NA mass values. It sorts based on the first column. If you switch mass and height, it sorts by mass in ascending order. If you get rid of second arg, it doesn't change the order.

Confirm your understanding by sorting using mass first then height.

```
# Write your code here.

newwars %>% arrange(mass, height) %>% head(20)
```

```
## # A tibble: 20 x 15
##    name     height  mass hair_color skin_color eye_color birth_year sex   gender
##    <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
##  1 Ratts T~     79  15   none       grey, blue unknown           NA male  mascu~
##  2 Yoda         66  17   white      green      brown            896 male  mascu~
##  3 Wicket ~     88  20   brown      brown      brown              8 male  mascu~
##  4 R2-D2        96  32   <NA>       white, bl~ red               33 none  mascu~
##  5 R5-D4        97  32   <NA>       white, red red               NA none  mascu~
##  6 Sebulba     112  40   none       grey, red  orange            NA male  mascu~
##  7 Dud Bolt     94  45   none       blue, grey yellow            NA male  mascu~
##  8 Padmé A~    185  45   brown      light      brown             46 fema~ femin~
##  9 Sly Moo~    178  48   none       pale       white             NA <NA>  <NA>
## 10 Wat Tam~    193  48   none       green, gr~ unknown           NA male  mascu~
## 11 Leia Or~    150  49   brown      light      brown             19 fema~ femin~
## 12 Barriss~    166  50   black      yellow     blue              40 fema~ femin~
## 13 Adi Gal~    184  50   none       dark       blue              NA fema~ femin~
## 14 Zam Wes~    168  55   blonde     fair, gre~ yellow            NA fema~ femin~
## 15 Ayla Se~    178  55   none       blue       hazel             48 fema~ femin~
## 16 Luminar~    170  56.2 black      yellow     blue              58 fema~ femin~
## 17 Shaak Ti    178  57   none       red, blue~ black             NA fema~ femin~
```

```
## 18 Ben Qua~    163  65   none       grey, gre~ orange            NA male  mascu~
## 19 Jar Jar~    196  66   none       orange     orange            52 male  mascu~
## 20 Nien Nu~    160  68   none       grey       black             NA male  mascu~
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

There is also a `top_n` function that is useful when we want to take a quick look at the top n values of the data without fully rearranging the rows.

```
newwars %>% top_n(5,avHW)
```

```
## # A tibble: 5 x 15
##   name      height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
## 1 Chewbacca    228   112 brown      unknown    blue             200 male  mascu~
## 2 Jabba De~    175  1358 <NA>       green-tan~ orange           600 herm~ mascu~
## 3 IG-88        200   140 none       metal      red               15 none  mascu~
## 4 Grievous     216   159 none       brown, wh~ green, y~         NA male  mascu~
## 5 Tarfful      234   136 brown      brown      blue              NA male  mascu~
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

What do you think occurs if we use a negative number for n? Try it out.

Use your new skills to determine the 10 oldest characters in the starwars dataset.

```
# Write your code here.
newwars %>% top_n(10, birth_year)
```

```
## # A tibble: 10 x 15
##    name     height  mass hair_color skin_color eye_color birth_year sex   gender
##    <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
##  1 C-3PO       167    75 <NA>       gold       yellow           112 none  mascu~
##  2 Chewbac~    228   112 brown      unknown    blue             200 male  mascu~
##  3 Jabba D~    175  1358 <NA>       green-tan~ orange           600 herm~ mascu~
##  4 Yoda         66    17 white      green      brown            896 male  mascu~
##  5 Palpati~    170    75 grey       pale       yellow            82 male  mascu~
##  6 Qui-Gon~    193    89 brown      fair       blue              92 male  mascu~
##  7 Finis V~    170    NA blond      fair       blue              91 male  mascu~
##  8 Ki-Adi-~    198    82 white      pale       yellow            92 male  mascu~
##  9 Cliegg ~    183    NA brown      fair       blue              82 male  mascu~
## 10 Dooku       193    80 white      fair       brown            102 male  mascu~
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

4.7 The `summarize` function is probably the most complicated of the `dplyr` functions we look at here. Its purpose is to calculate summaries (such as mean, median, standard deviation, etc) of the variables in a dataframe. By itself it doesn't do much; other functions do the actual work.

```
newwars %>% summarize(avgH=mean(height,na.rm=T),std1=sd(height,na.rm=T))
```

```
## # A tibble: 1 x 2
##    avgH  std1
##   <dbl> <dbl>
## 1  175.  34.8
```

(Note that we include the `na.rm` argument to strip NAs before the calculations.)

The output is a dataframe; in this case, the output has one row and two columns.

Describe in words what you output tells us.

How could you do this using base R functionality?

```
# Write your code here.
data.frame(avgH = mean(newwars$height, na.rm=T), std1 = sd(newwars$height, na.rm=T))
```

```
##       avgH     std1
## 1 174.6049 34.77416
```

There is a version of `summarize` called `summarize_all` that could save some time. It calculates summaries (mean, or median, or standard deviation, or others) of *all* the variables in a dataframe. If a summary is not appropriate, it returns `NA`.

```
newwars %>% summarize_all(mean, na.rm=T)
```

```
## Warning: There were 11 warnings in ‘summarise()‘.
## The first warning was:
## i In argument: ‘name = (function (x, ...) ...‘.
## Caused by warning in ‘mean.default()‘:
## ! argument is not numeric or logical: returning NA
## i Run ‘dplyr::last_dplyr_warnings()‘ to see the 10 remaining warnings.
```

```
## # A tibble: 1 x 15
##    name height  mass hair_color skin_color eye_color birth_year   sex gender
##   <dbl>  <dbl> <dbl>      <dbl>      <dbl>     <dbl>      <dbl> <dbl>  <dbl>
## 1    NA   175.  97.3         NA         NA        NA       87.6    NA     NA
## # i 6 more variables: homeworld <dbl>, species <dbl>, films <dbl>,
## #   vehicles <dbl>, starships <dbl>, avHW <dbl>
```

You can use `summarize_if` in order to avoid getting NAs.

```
newwars %>% summarize_if(is.numeric, mean, na.rm=T)
```

```
## # A tibble: 1 x 4
##   height  mass birth_year  avHW
##    <dbl> <dbl>      <dbl> <dbl>
## 1   175.  97.3       87.6  136.
```

Use the summarize function to make a new dataframe that is composed of the mean and standard deviation of `height` and `mass` in the `newwars` dataset.

```
# Write code here.
newwars %>% summarize(avgH=mean(height,na.rm=T),std1=sd(height,na.rm=T), avgM = mean(mass, na.rm=T), st
```

```
## # A tibble: 1 x 4
##    avgH  std1  avgM  std2
##   <dbl> <dbl> <dbl> <dbl>
## 1  175.  34.8  97.3  169.
```

But the real usefulness of `summarize` is when you combine it with the function `group_by`. When applied to a dataframe, the `group_by` function produces an object called a *grouped dataframe*, which is pretty similar to an ordinary dataframe, except it is aware that the data is divided into several groups.

Groups are determined by categorical variables. Let's apply the `group_by` function to the newwars.

```
newwars$species <- as.factor(newwars$species)
groupedwars <- newwars %>% group_by(species)
head(groupedwars)
```

```
## # A tibble: 6 x 15
## # Groups:   species [2]
##   name      height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
## 1 Luke Sky~    172    77 blond      fair       blue            19   male  mascu~
## 2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu~
## 3 R2-D2         96    32 <NA>       white, bl~ red             33   none  mascu~
## 4 Darth Va~    202   136 none       white      yellow          41.9 male  mascu~
## 5 Leia Org~    150    49 brown      light      brown           19   fema~ femin~
## 6 Owen Lars    178   120 brown, gr~ light      blue            52   male  mascu~
## # i 6 more variables: homeworld <chr>, species <fct>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>
```

Not much different from just the `newwars` dataframe. It just identifies the groups and changes how it acts with other functions. If we apply the `summarize` function to this new dataframe, using the `height`, we get the following:

```
summarize(groupedwars,avg=mean(height, na.rm=T))
```

```
## # A tibble: 38 x 2
##    species     avg
##    <fct>     <dbl>
##  1 Aleena      79
##  2 Besalisk   198
##  3 Cerean     198
##  4 Chagrian   196
##  5 Clawdite   168
##  6 Droid      131.
##  7 Dug        112
##  8 Ewok        88
##  9 Geonosian  183
## 10 Gungan     209.
## # i 28 more rows
```

We get the mean of the height of each group. Compare that with what we get if the dataframe is not grouped:

```
summarise(newwars, avg=mean(height, na.rm=T))
```

```
## # A tibble: 1 x 1
##     avg
##   <dbl>
## 1  175.
```

We get the mean height of *all* the characters in the dataset.

A final note: we can get the means of all the groups with one line of code if we use pipes.

```
newwars %>% group_by(species) %>% summarize(avg=mean(height,na.rm=T))
```

```
## # A tibble: 38 x 2
##    species     avg
##    <fct>     <dbl>
##  1 Aleena      79
##  2 Besalisk   198
##  3 Cerean     198
##  4 Chagrian   196
##  5 Clawdite   168
##  6 Droid      131.
##  7 Dug        112
##  8 Ewok        88
##  9 Geonosian  183
## 10 Gungan     209.
## # i 28 more rows
```

For the purposes of readability, I actually prefer to put each step of the process on different lines, as below:

```
newwars %>%
  group_by(species) %>%
  summarize(avg=mean(height,na.rm=T))
```

```
## # A tibble: 38 x 2
##    species     avg
##    <fct>     <dbl>
##  1 Aleena      79
##  2 Besalisk   198
##  3 Cerean     198
##  4 Chagrian   196
##  5 Clawdite   168
##  6 Droid      131.
##  7 Dug        112
##  8 Ewok        88
##  9 Geonosian  183
## 10 Gungan     209.
## # i 28 more rows
```

Below is another example. Explain in your own words what it does in a step-by-step manner.

A: newwars is being accessed so then it can use the group_by() function to group the homeworld variable. This new dataframe is accessed by the summarize function in order to find the standard deviations of the heights. These are then assigned to newwars2 in order to create a dataframe with the homeworlds and the standard deviations

```
newwars2 <- newwars %>%
  group_by(homeworld) %>%
  summarize(sd=sd(height,na.rm=T))
newwars2
```

```
## # A tibble: 49 x 2
##    homeworld         sd
##    <chr>          <dbl>
##  1 Alderaan       22.9
##  2 Aleen Minor    NA
##  3 Bespin         NA
##  4 Bestine IV     NA
##  5 Cato Neimoidia NA
##  6 Cerea          NA
##  7 Champala       NA
##  8 Chandrila      NA
##  9 Concord Dawn   NA
## 10 Corellia        7.07
## # i 39 more rows
```

Let's use the famous `iris` dataset to get a little more practice. It has measurements of lengths and widths of three kinds of irises: setosa, versicolor, and virginica.

Link: Iris wikipedia

Calculate the average `Petal.Width` by `Species` *and* put the values in DESCENDING order by the average. Do all of this in one series of pipes.

```
# Write your code here.
iris %>%
  group_by(Species) %>%
  summarize(AvgPW = mean(Petal.Width, na.rm=T)) %>%
  arrange(desc(AvgPW))
```

```
## # A tibble: 3 x 2
##   Species    AvgPW
##   <fct>      <dbl>
## 1 virginica  2.03
## 2 versicolor 1.33
## 3 setosa     0.246
```

Here is another challenge. Can you do the SAME thing I asked you to do with the iris data but with base R functionality? Give it your best attempt.

```
# Write your code here.
species <- unique(iris$Species)
mean1 <- mean(iris$Petal.Width[iris$Species == species[1]])
```

```
mean2 <- mean(iris$Petal.Width[iris$Species == species[2]])
mean3 <- mean(iris$Petal.Width[iris$Species == species[3]])

AvgPW <- c(mean1, mean2, mean3)
Species <- rev(species)
AvgPW <- rev(AvgPW)

Petals <- data.frame(Species, AvgPW)
Petals
```

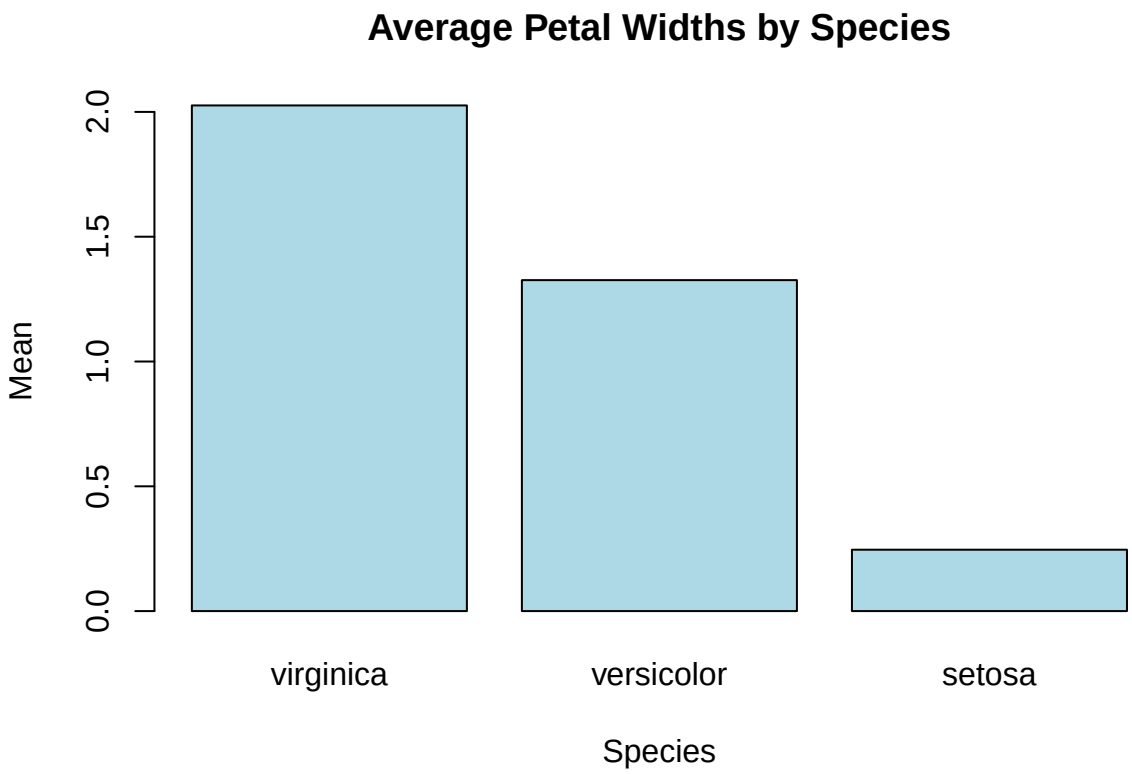
```
##      Species AvgPW
## 1  virginica 2.026
## 2 versicolor 1.326
## 3     setosa 0.246
```

What do you think of piping? Do you find it intuitive and useful, or are you annoyed that you have another thing to learn?

A: I do find it a lot more intuitive. Trying to find a way to do the same things with base R functionality is much harder and you have to got through a lot more steps. The syntax also makes more sense.

OK, so we have all this nice wrangled data. What is something we can do with it? Make a plot! For the mean `Petal.Width` by `Species` data we have a quantitative numeric variable (`Petal.Width`) and a categorical variable (`Species`). This means we make a barplot. Explore the `barplot` help documentation and make the appropriate barplot.

```
# Write your code here.

#help('barplot')

barplot(Petals$AvgPW,
        main = "Average Petal Widths by Species",
        xlab = "Species",
        ylab = "Mean",
        names.arg = Petals$Species,
        col = "lightblue",
        border = "black")
```



4.11 Section 4.11 discusses *tibbles*, which are opinionated versions of dataframes. The `tidyverse` package prefers tibbles over ordinary dataframes, for reasons listed in section 4.11.

Read this section and explain which of the reasons given is the most compelling to you.

A: I think the versatility of it as specified in 4.11.2 is probably the most compelling reason to me. It can return a tibble, but if you want the vector/column you can still use the $ to access it. It is important that it returns a tibble when you subset because some functions require you to input a data frame which can only be done this way with tidyverse, but if you use a function requires a vector, you can just as easily access it. This allows for more functions to be used and allows for more ability to work with a data frame.

4.8 and 4.12 Read about the `pull` function and the placeholder operator in sections 4.8 and 4.12. We will use them as needed.

Write some example code that illustrates the placeholder operator. Your example MUST be different than the one given in the text.

```
#Write your code here.

4 %>% sqrt(x = .)
```

```
## [1] 2
```

```
c(1, 2, 3, 4, 5) %>% max(.)
```

```
## [1] 5
```

```
c(1, 2, 3, 4, 5) %>% min(.)
```

```
## [1] 1
```

```
c(1, 2, 3, 4, 5) %>% mean(.)
```

```
## [1] 3
```

4.13 The `purrr` package is described in section 4.13. Within the purrr function is the `map` function, which does similar things to `sapply`.

`sapply` applies functions over a list, vector, matrix, or dataframe. For example, we can use it if we want the mean for all the quantitave variables in our `newwars` dataframe:

```
sapply(newwars, mean, na.rm=T)
```

```
##       name     height       mass hair_color skin_color  eye_color birth_year
##         NA 174.60494   97.31186         NA         NA         NA   87.56512
##        sex     gender  homeworld    species      films   vehicles  starships
##         NA         NA         NA         NA         NA         NA         NA
##       avHW
##  135.83390
```

`map` does basically the same thing, but is preferred over `sapply` because it *always* returns a list. Whereas, what `sapply` returns depends on what you give it.

```
map(newwars, mean, na.rm=T)
```

```
## $name
## [1] NA
##
## $height
## [1] 174.6049
##
## $mass
## [1] 97.31186
##
## $hair_color
## [1] NA
##
## $skin_color
## [1] NA
##
## $eye_color
## [1] NA
##
## $birth_year
## [1] 87.56512
##
## $sex
## [1] NA
```

```
##
## $gender
## [1] NA
##
## $homeworld
## [1] NA
##
## $species
## [1] NA
##
## $films
## [1] NA
##
## $vehicles
## [1] NA
##
## $starships
## [1] NA
##
## $avHW
## [1] 135.8339
```

The `map_dbl` function always returns a vector of numerical values.

```
map_dbl(newwars, mean, na.rm=T)
```

```
##       name     height       mass hair_color skin_color  eye_color birth_year
##         NA  174.60494   97.31186         NA         NA         NA   87.56512
##        sex     gender  homeworld    species      films   vehicles  starships
##         NA         NA         NA         NA         NA         NA         NA
##       avHW
##  135.83390
```

Use both `sapply` and `map_dbl` to produce the means of all the numerical variables in the `iris` dataset. Do the results look similar to each other?

```
# Write your code here.
sapply(iris, mean, na.rm=T)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##     5.843333     3.057333     3.758000     1.199333           NA
```

```
map_dbl(iris, mean, na.rm=T)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##     5.843333     3.057333     3.758000     1.199333           NA
```

A: The results look identical.

4.14 Often we need to define a new categorical variable based on the values of some other variable. We do this by using use the `case_when` and `between` operators.

The `between` function determines if a value falls within an interval. So we can ask the hard hitting questions like is Luke Skywalker's height between 1.5 and 2 meters?

```
between(newwars$height[newwars$name=="Luke Skywalker"],150,200)
```

```
## [1] TRUE
```

Using `between` is certainly easier than relying only on the `>` and `<` operators. Reproduce the above between code using the `>` and `<` operators.

```
newwars$height[newwars$name=="Luke Skywalker"] > 150 & newwars$height[newwars$name=="Luke Skywalker"] <
```

```
## [1] TRUE
```

Now use the `between` function to determine if Darth Vader's height is between that of Luke Skywalker and Chewbacca.

```
#Write your code here.
between(newwars$height[newwars$name == "Darth Vader"],
        newwars$height[newwars$name == "Luke Skywalker"],
        newwars$height[newwars$name == "Chewbacca"])
```

```
## [1] TRUE
```

```
#newwars$height[newwars$name == "Darth Vader"] --> 202
#newwars$height[newwars$name == "Luke Skywalker"] --> 172
#newwars$height[newwars$name == "Chewbacca"] --> 228
```

The `case_when` function is useful for vectorizing conditional statements. It is similar to `ifelse` but can output any number of values, as opposed to just TRUE or FALSE.

```
case_when(
  between(newwars$height[newwars$name=="Luke Skywalker"],150,200) ~ "Luke Skywalker is between 150 and 
  TRUE ~ "Luke Skywalker is not between 150 and 200 cm tall")
```

```
## [1] "Luke Skywalker is between 150 and 200 cm tall."
```

Now let's get to the part where we define a new categorical variable based on the values of some other variable. Let's create a categorical height variable for the star wars characters.

```
newwars3 <- newwars %>% mutate(catheight = case_when (
  between(height,0,99) ~ "Short",
  between(height,100,199) ~ "Medium",
  between(height,200,299) ~ "Tall",
  TRUE ~ "F"
))

head(newwars3)
```

```
## # A tibble: 6 x 16
##   name      height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
```

```
## 1 Luke Sky~    172    77 blond      fair       blue            19   male  mascu~
## 2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu~
## 3 R2-D2         96    32 <NA>       white, bl~ red             33   none  mascu~
## 4 Darth Va~    202   136 none       white      yellow          41.9 male  mascu~
## 5 Leia Org~    150    49 brown      light      brown           19   fema~ femin~
## 6 Owen Lars    178   120 brown, gr~ light      blue            52   male  mascu~
## # i 7 more variables: homeworld <chr>, species <fct>, films <list>,
## #   vehicles <list>, starships <list>, avHW <dbl>, catheight <chr>
```

Now it is your turn. For the exams dataset, create a categorical grade variable for exam1 where 45-50 is an "A", 40-44 is a "B", 35-39 is a "C", and 30-34 is a "D".

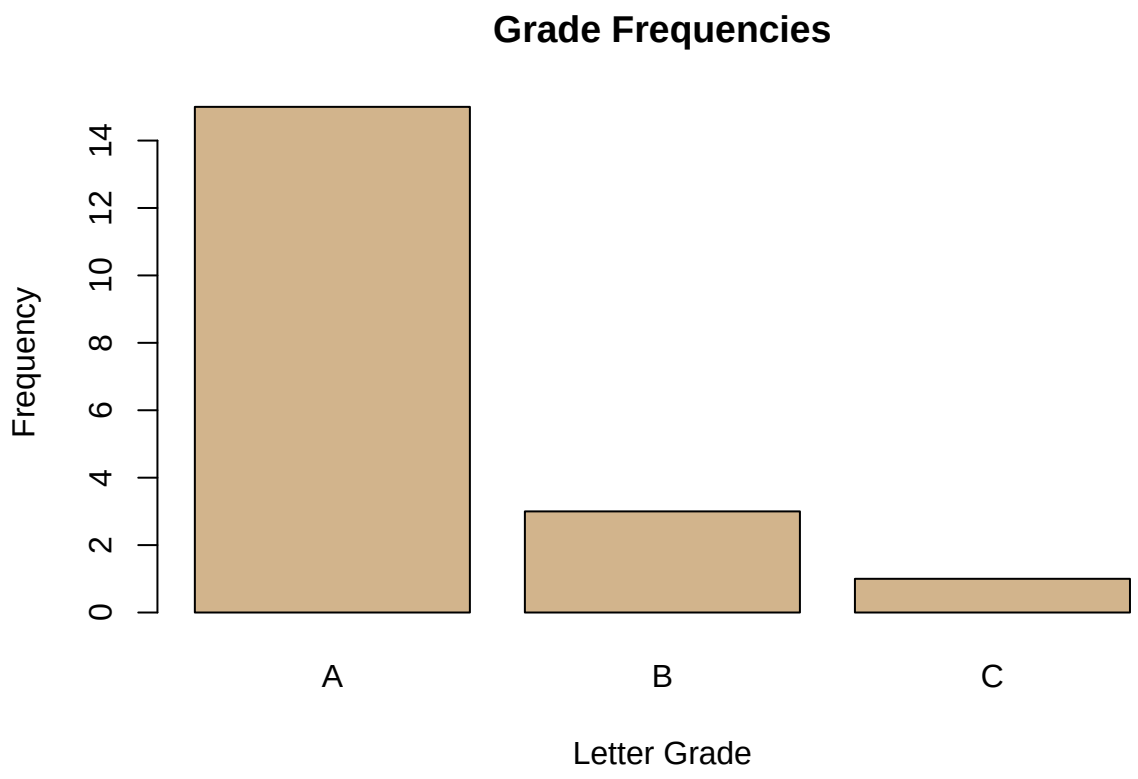
```
# Write your code here.
load('exams.rda')
exams3 <- exams %>% mutate(catgrade = case_when(
  between(exam1,45,50) ~ "A",
  between(exam1,40,44) ~ "B",
  between(exam1,35,39) ~ "C",
  between(exam1,30,34) ~ "D",
  TRUE ~ "F"
))

exams3
```

```
##    ID exam1 exam2 improve catgrade
## 1   1    48    48       0        A
## 2   2    47    46      -1        A
## 3   3    39    42       3        C
## 4   4    46    47       1        A
## 5   5    45    45       0        A
## 6   6    50    50       0        A
## 7   7    50    45      -5        A
## 8   8    41    43       2        B
## 9   9    50    46      -4        A
## 10 10    47    46      -1        A
## 11 11    41    39      -2        B
## 12 12    50    50       0        A
## 13 13    47    47       0        A
## 14 14    47    45      -2        A
## 15 15    47    31     -16        A
## 16 16    42    31     -11        B
## 17 17    47    49       2        A
## 18 18    46    45      -1        A
## 19 19    46    39      -7        A
```

Bonus: can you figure out how to make a barplot showing the frequencies of each letter grade?

```
# Write your code here.
count <- table(exams3$catgrade)
grade <- unique(exams3$catgrade)
barplot(count,
        main = "Grade Frequencies",
        xlab = "Letter Grade",
```

```
        ylab = "Frequency",
        col = "tan")
```



If you are interested in even more practice, work through the exercises in sections 4.6, 4.10, and 4.15 of the textbook:

rafalab.dfci.harvard.edu/dsbook/tidyverse.html#exercises-10

rafalab.dfci.harvard.edu/dsbook/tidyverse.html#exercises-11

rafalab.dfci.harvard.edu/dsbook/tidyverse.html#exercises-12